

VISIÓN POR COMPUTADOR - LABORATORIO 03

Ejercicio 1:

En el primer ejercicio se solicitaba una función que aplicase un filtro de media aritmética (a un píxel respecto a sus vecinos) de forma eficiente. Para ello nosotros hemos aplicado el filtro primero verticalmente, y después horizontalmente para agilizar los accesos a las matrices. El código de la función ejercicio_1 es el siguiente:

```
function F = Ejercicio_1(A)

    ones_h = ones(1,3)/3;
    ones_v = ones_h';

    F = imfilter(A, ones_h);
    F = imfilter(F, ones_v);

end
```

Como lo separamos en la aplicación de dos filtros, vertical y horizontal, creamos dos matrices de dimensión 1x3 y 3x1 respectivamente.

Sobre la imagen original aplicamos el filtro horizontal, y posteriormente sobre la imagen filtrada horizontalmente, aplicamos el filtro vertical.

El resultado es una imagen con un efecto blurring:

Input:



Output:



Ejercicio 2:

El segundo ejercicio pide una función que actúe como filtro no lineal para eliminar el ruido de tipo 'salt & pepper' de una imagen en blanco y negro. Dicho filtro está basado en el cálculo de las medianas de los vecinos para sustituir los píxeles 'sospechosos' de ser ruidosos.

Este filtro se aplica a través de la función `colfilt`, en modo `sliding`, que va progresivamente llamando a nuestra función `Ejercicio_2`, que es la encargada de computar el filtro.

```
F = colfilt(I, [9 9], 'sliding', @Ejercicio_2);
```

```
function promig = Ejercicio_2(A)
    % Paso 0
    [f, c] = size(A);
    c = double(c);
    % Paso 1
    original = A(1:uint32(f)/2,:);
    % Paso 2
    zero = zeros(1,c);
    black_pixels = xor(original,zero);
    white_pixels = xor((original-255),zero);
    % Paso 3
    noise = or(not(black_pixels), not(white_pixels));
    % Paso 4
    not_noise = not(noise);
    % Paso 5
    promig = not_noise.*original + noise.*median(A);
end
```

- Paso 0: Calculamos el número de columnas que tenemos que tratar.
- Paso 1: En la variable `original` guardamos los píxeles centrales.
- Paso 2: Para separar los píxeles ruidosos (negros y blancos) del resto.
 - **Negros:** realizamos una operación XOR entre los originales y un vector completo de ceros. Obtenemos como resultado un vector lleno de unos excepto allí donde hay un pixel negro.
 - **Blancos:** Siguiendo el mismo razonamiento, a los píxeles originales les restamos el color blanco (255) poniendo a si los píxeles blancos a 0 para poder aplicar la misma operación que para los negros.
- Paso 3: Negamos los vectores, obteniendo como resultado, vectores con 1 en las posiciones donde hay píxeles que pertenecen al ruido. A su vez, aplicamos una OR para unificar ambos conjuntos en un solo vector, el que recibe el nombre `noise` (píxeles que forman parte del ruido = 1).

- Paso 4: De nuevo, negamos el vector de píxeles ruidosos, obteniendo así el vector de píxeles no ruidosos.
- Paso 5: Finalmente, utilizamos los anteriores vectores a modo de máscara, aplicando la operación de la media a los píxeles ruidosos y dejando los píxeles originales en caso de no pertenecer al ruido. Una de las dos partes de la suma siempre será 0, ya que los vectores máscara son complementarios.

Input:



Output:



Ejercicio 3:

El ejercicio 3 consiste en realzar los contornos de la imagen ponderándola con la imagen gradiente. Inicialmente debemos pasar la imagen a nivel de grises para poder aplicar un filtro gaussiano para suavizar la imagen y posteriormente obtener el gradiente mediante sobel.

```
I=rgb2gray(I);  
I=imgaussfilt(I,0.6);  
  
[Gmag, Gdir] = imgradient(I,'sobel');
```

Finalmente binarizamos la imagen original usando la magnitud del gradiente obtenido con un umbral de 100.

```
for i= 1:size(I,1)  
    for j=1:size(I,2)  
        if Gmag(i,j) > 100  
            I(i,j)=255;  
        else  
            I(i,j)=0;  
        end  
    end  
end
```

Input:



Output:



Ejercicio 4:

Para este ejercicio vamos a realizar una función para que simule un motion blur como parámetros de entrada recibe el path, un tamaño para el filtro y finalmente el ángulo que queremos que simula, en este ejemplo hemos utilizado la imagen anterior, un size de 35 y un ángulo de 45 grados.

Primero se crea una matriz H de dimensiones "size" y se inicializa a cero y acto seguido se pone a unos la fila central y se divide para que la ponderación final sea uno.

```
H = zeros(size);  
H(int32(floor(size)/2),1:size) = 1;  
H = H/ceil(size);
```

Para aplicar el efecto de motion blur requerimos de rotar la matriz H y al final aplicar el filtro sobre la imagen original.

```
H = imrotate(H,alpha);
```

```
B = imfilter(I,H);
```

Input:



Output:

